Data Networks
UdS and IMPRS-CS

Lecture 16: Quality of Service (QoS)

Recap: TCP Congestion Control

- Measure available bandwidth
  - Slow start: fast, hard on network
  - Congestion avoidance: AIMD is slow, gentle on network
- Detecting congestion
  - timeout based on RTT; does not permit fast recovery
    - robust, causes low throughput
  - Fast Retransmit: avoids timeouts when few packets lost
    - can be fooled, maintains high throughput
  - Fast recovery: don't set cwnd=1 with fast retransmits

Recap: Issues with TCP congestion control

- Increase rate until packet loss
  - Drives network into congestion; high queuing delay, inefficient utilization
  - Solution: Delay-based congestion control, RED, ECN
- Use loss as indication of congestion
  - Cannot distinguish congestion from packet corruption
  - Solution: ECN
- AIMD mechanism oscillates around proper rate
  - Rate is not smooth: Bad for streaming applications (e.g. video)
  - Solution: Equation-based congestion control
- Slow start to probe for initial rate
  - Bad for short lived flows (e.g. most Web transfers, a lot of Internet traffic is web transfer)
  - Solution: Probe-gap based congestion control
- Relies on AIMD behavior of end hosts for fairness
  - People can cheat (not use AIMD)
  - People can open many parallel connections
  - Solution: Router-based active queue management -- today's discussion

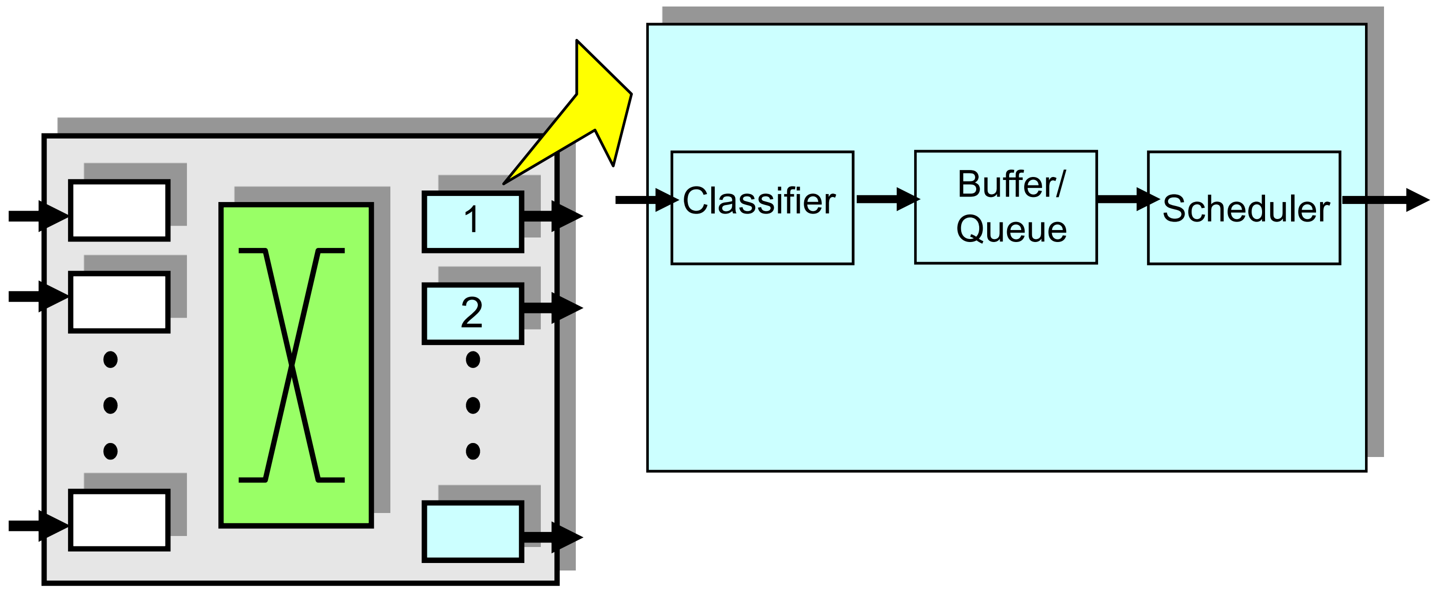
What Can a Basic Router do to Packets?

- Send it…
- Delay it…
- Drop it…
- How they are done impacts Quality of Service
  - Best effort? Guaranteed delay? Guaranteed throughput?

- Many variations in policies with different behavior
- Rich body of research work to understand them
- Limited Internet deployment
  - Many practical deployment barriers since Internet was best-effort to begin with, adding new stuff is hard
  - Some people just don’t believe in the need for QoS! Not enough universal support

Router Architecture Assumptions

- Assumes inputs just forward packets to outputs
  - Switch core is N times faster than links in a NxN switch
  - No contention at input, no head-of-line blocking
  - Remember homework 2?
- Resource contention occurs only at the output interfaces
- Output interface has classifier, buffer/queue, scheduler components

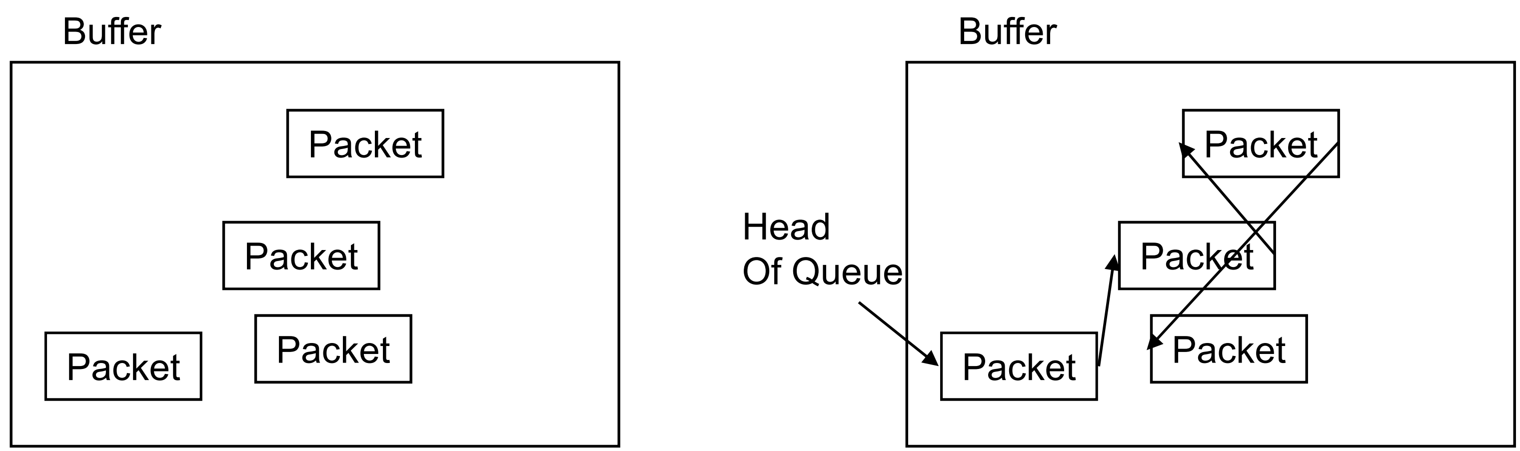


Internet Classifier

- A "flow" is a sequence of packets that are related (e.g. from the same application)
- Flow in Internet can be identified by a subset of following fields in the packet header
  - source/destination IP address (32 bits)
  - source/destination port number (16 bits)
  - protocol type (8 bits)
  - type of service (4 bits)
- Examples:
  - All TCP packets from Krishna's web browser on machine A to web server on machine B
  - All packets from MPI-SWS
  - All packets between MPI-SWS and Saarland
  - All UDP packets from CS department
- Classifier takes a packet and decides to which flow it belongs

Buffer/Queue

- Buffer: memory where packets can be stored temporarily
- Queue: using buffers to store packets in an ordered sequence
  - E.g. First-in-First-Out (FIFO) queue



Buffer/Queue

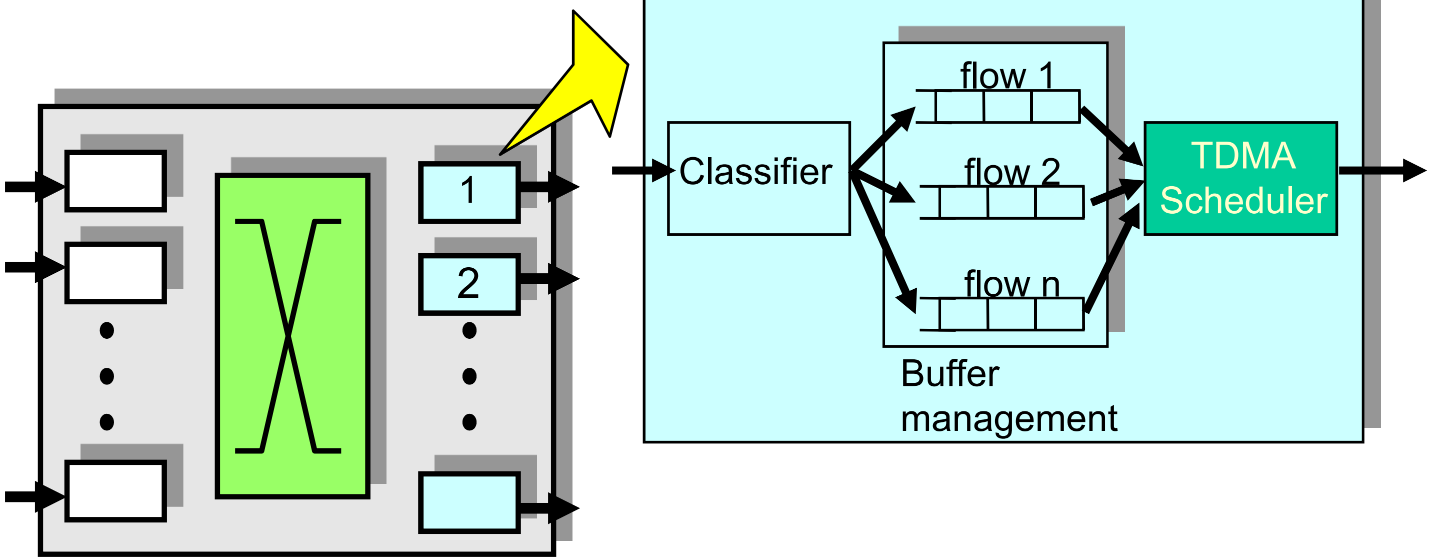
- When packets arrive at an output port faster than the output link speed (perhaps only momentarily)
- Can drop all excess packets
  – Resulting in terrible performance
- Or can hold excess packets in buffer/queue
  – Resulting in some delay, but better performance
- Still have to drop packets when buffer is full
  – For a FIFO queue, “drop tail” or “drop head” are common policies
  – i.e. drop last packet to arrive vs drop first packet in queue to make room

- A chance to be smart: Transmission of packets held in buffer/queue can be *scheduled*
  – Which stored packet goes out next? Which is more “important”?
  – Impacts quality of service

Scheduler

- Decides how the output link capacity is shared by flows
  - Which packet from which flow gets to go out next?
- E.g. FIFO schedule
  - Simple schedule: whichever packet arrives first leaves first
  - Agnostic of concept of flows, no need for classifier, no need for a real “scheduler”, a FIFO queue is all you need
- E.g. TDMA schedule
  - Queue packets according to flows
    - Need classifier and multiple FIFO queues
  - Divide transmission times into slots, one slot per flow
  - Transmit a packet from a flow during its time slot

TDMA Example



Internet Today

- FIFO queues are used at most routers
- No classifier, no scheduler, best-effort

- Sophisticated mechanisms tend to be more common near the “edge” of the network
  - E.g. At campus routers
  - Use classifier to pick out Kazaa packets
  - Use scheduler to limit bandwidth consumed by Kazaa traffic

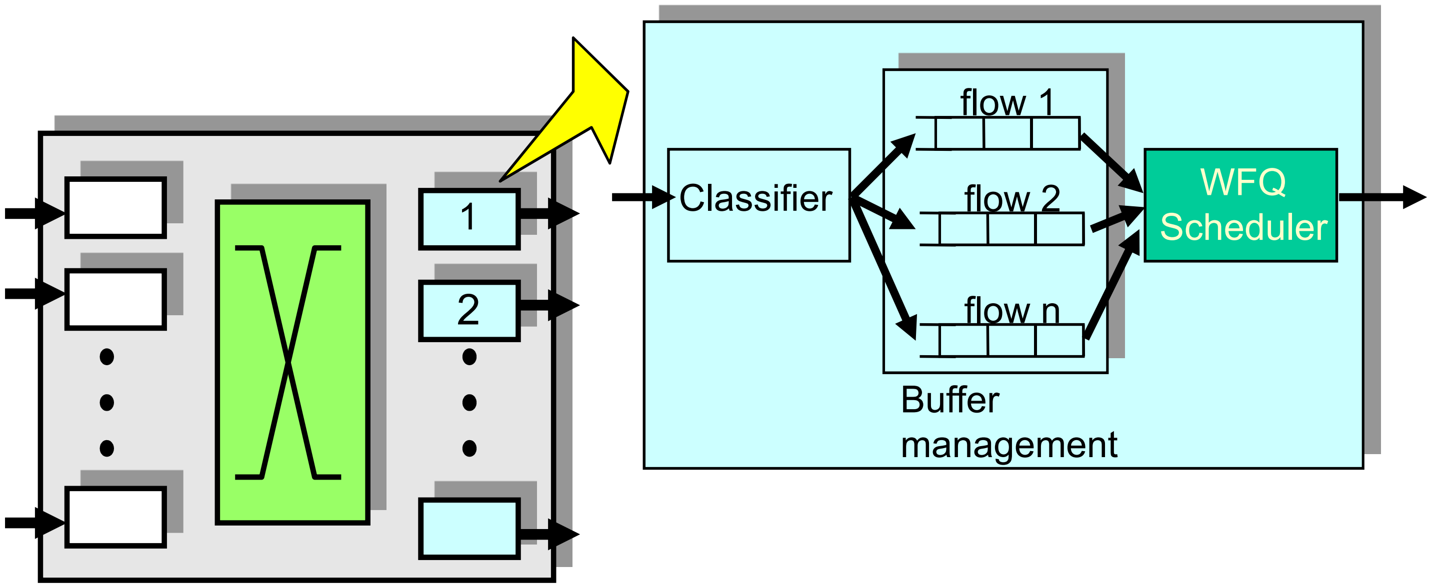
Achieving QoS in Statistical Multiplexing Network

- We want guaranteed QoS
- But we don't want the inefficiency of TDMA
  - Unused time slots are "wasted"

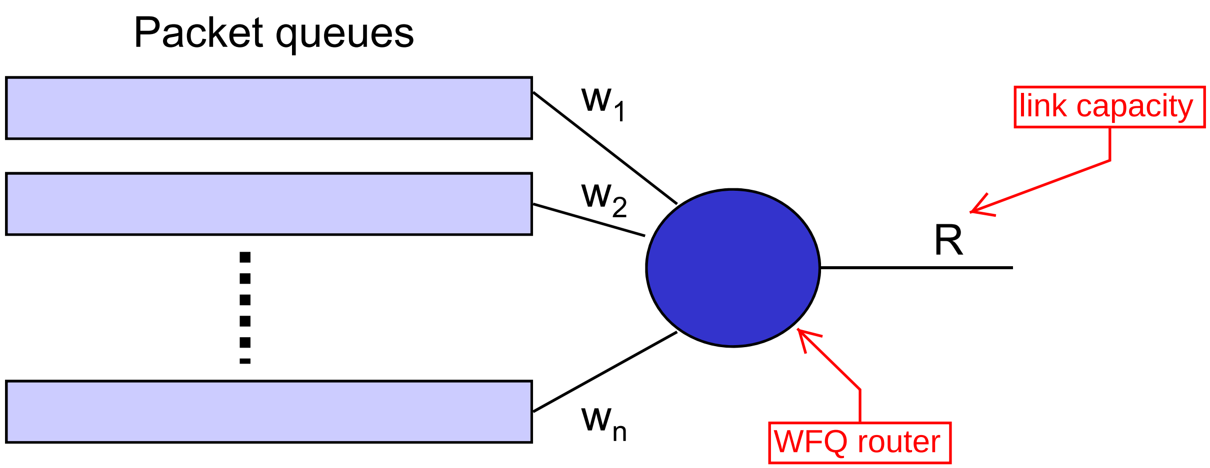
- Want to statistically share un-reserved capacity or reserved but unused capacity

- One solution: Weighted Fair Queuing (WFQ)
  - Guarantees a flow receives at least its allocated bit rate

WFQ Architecture



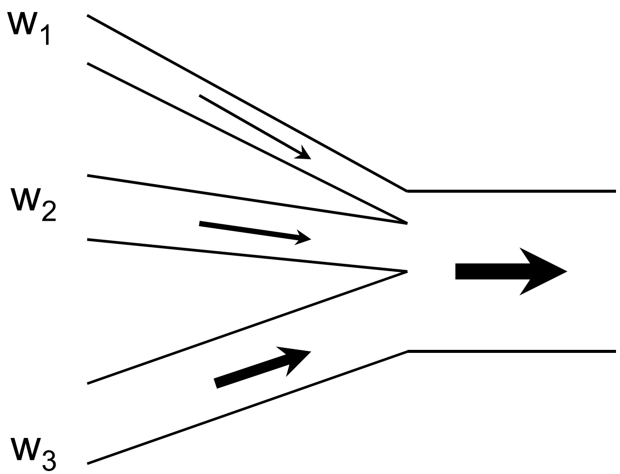
What is Weighted Fair Queueing?



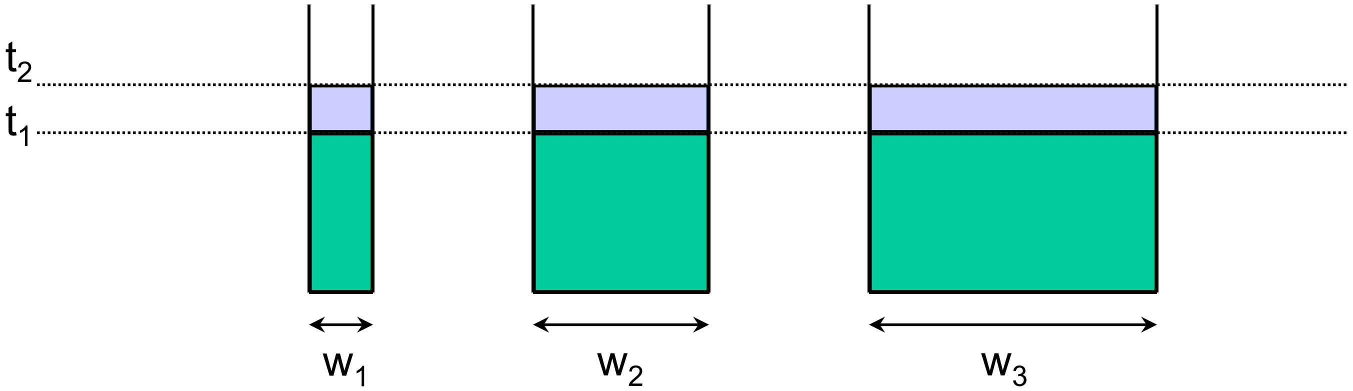
- Each flow i given a weight (importance) w_i
- WFQ guarantees a minimum service rate to flow i
  - $r_i = R * w_i / (w_1 + w_2 + ... + w_n)$
  - Implies isolation among flows (one cannot mess up another)

What is the Intuition? Fluid Flow

water pipes



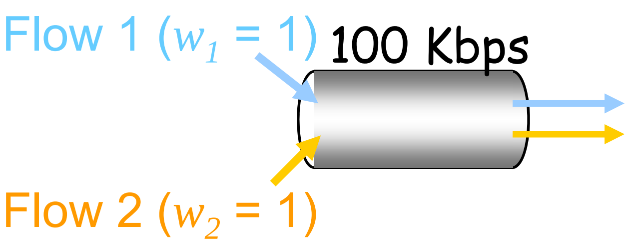
water buckets



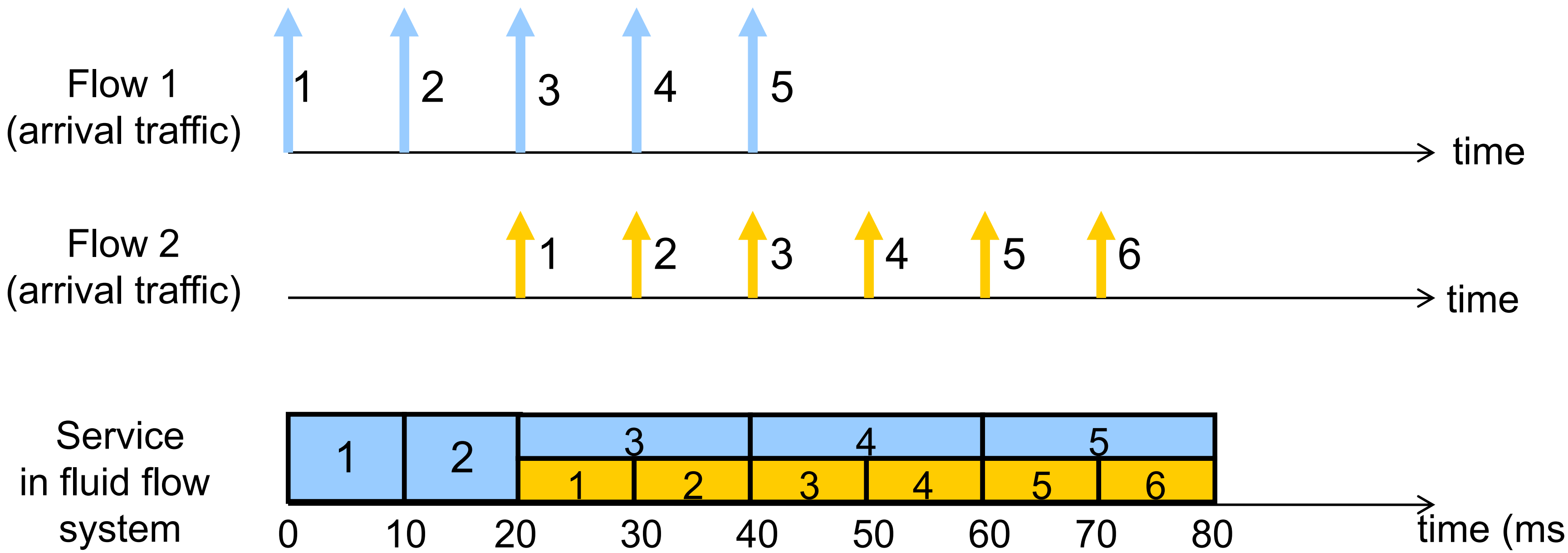
Fluid Flow System

- If flows can be served one bit at a time
- WFQ can be implemented using bit-by-bit weighted round robin
  - During each round from each flow that has data to send, send a number of bits equal to the flow's weight

Fluid Flow System: Example 1

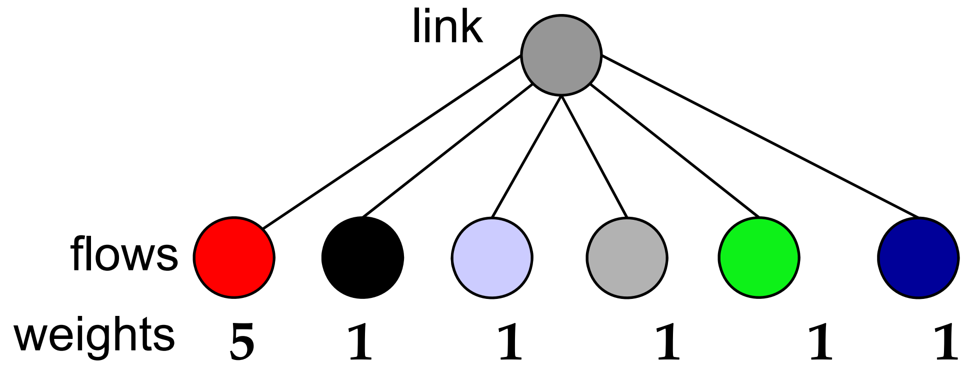


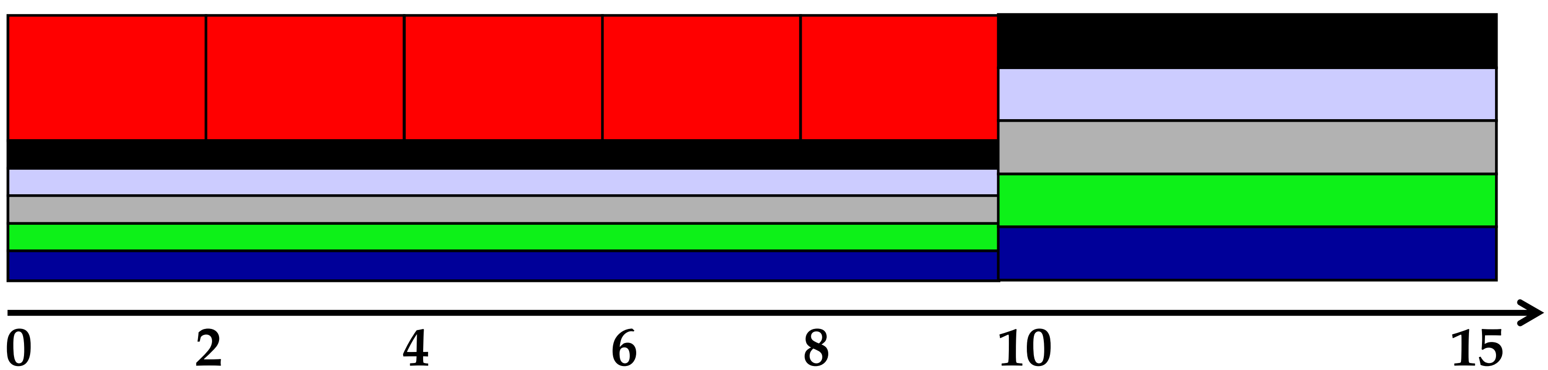
	Packet Size (bits)	Packet inter-arrival time (ms)	Arrival Rate (Kbps)
Flow 1	**1000**	**10**	**100**
Flow 2	**500**	**10**	**50**



Fluid Flow System: Example 2

- Red flow has packets backlogged between time 0 and 10
  - Backlogged flow → flow's queue not empty
- Other flows have packets continuously backlogged
- All packets have the same size

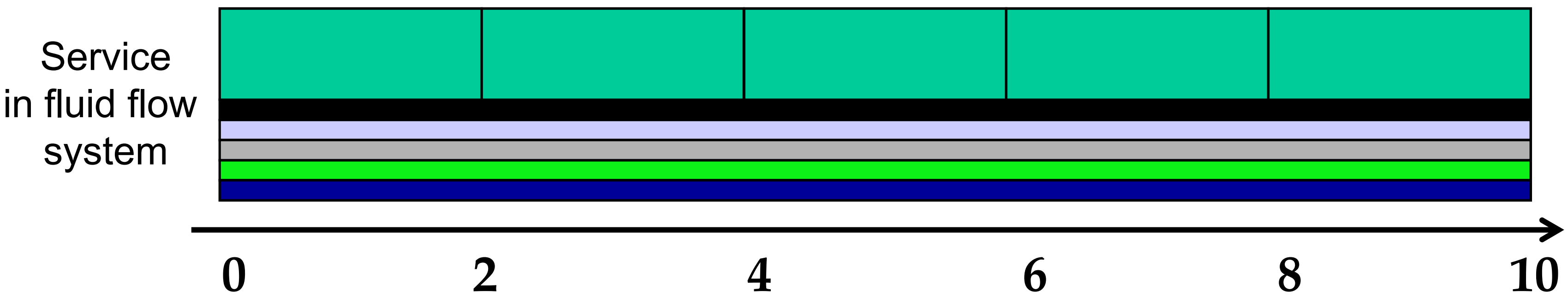




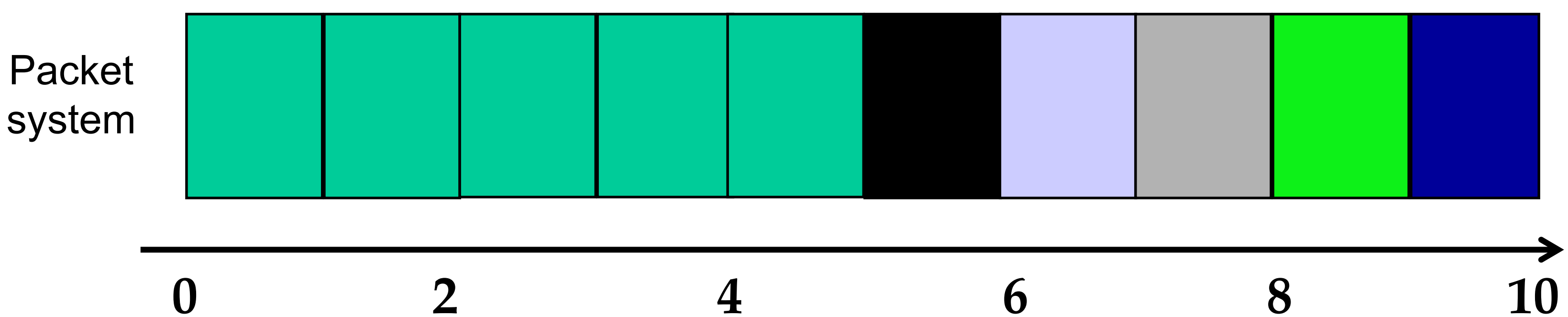
Implementation in Packet System

- Packet (Real) system: packet transmission cannot be preempted. Why? ← otherwise you have a packet corruption
- Solution: serve packets in the order in which they would have finished being transmitted in the fluid flow system

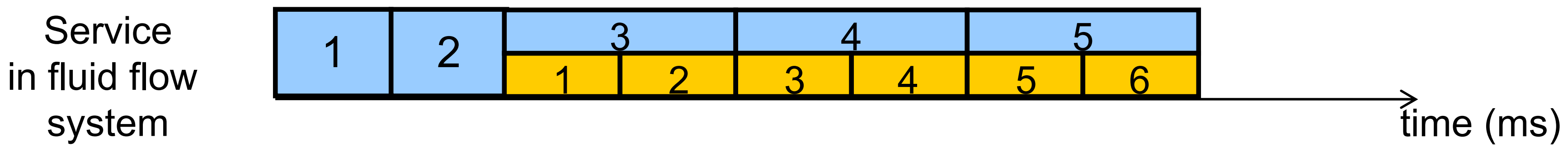
Packet System: Example 1



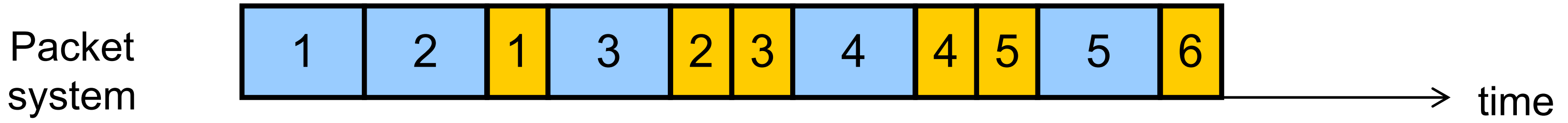
- Select the first packet that finishes in the fluid flow system



Packet System: Example 2



- Select the first packet that finishes in the fluid flow system

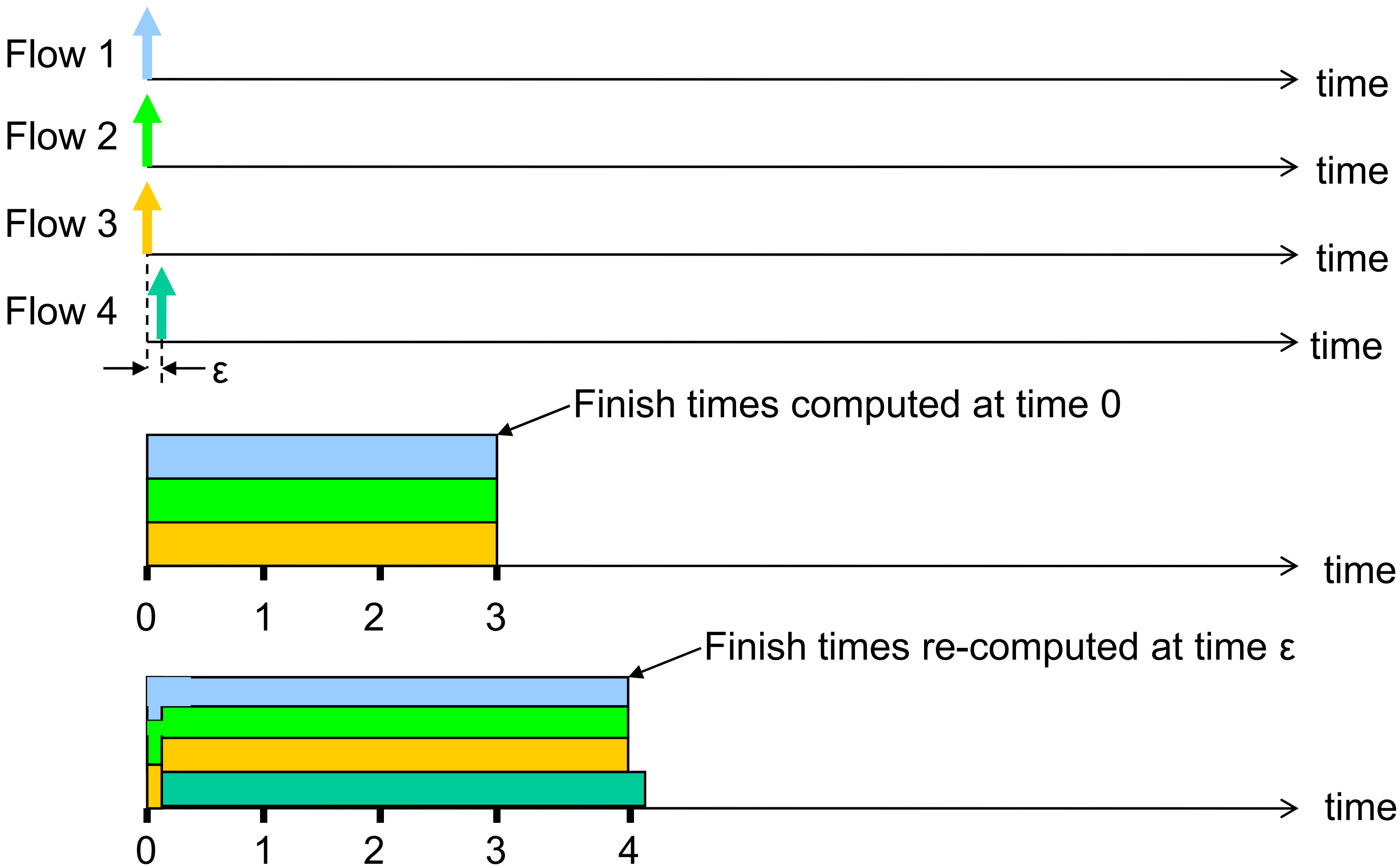


Implementation Challenge

- Need to compute the finish time of a packet in the fluid flow system…
- … but the finish time may change as new packets arrive!
- Need to update the finish times of all packets that are in service in the fluid flow system when a new packet arrives
  - But this is very expensive; a high speed router may need to handle hundred of thousands of flows!

Example

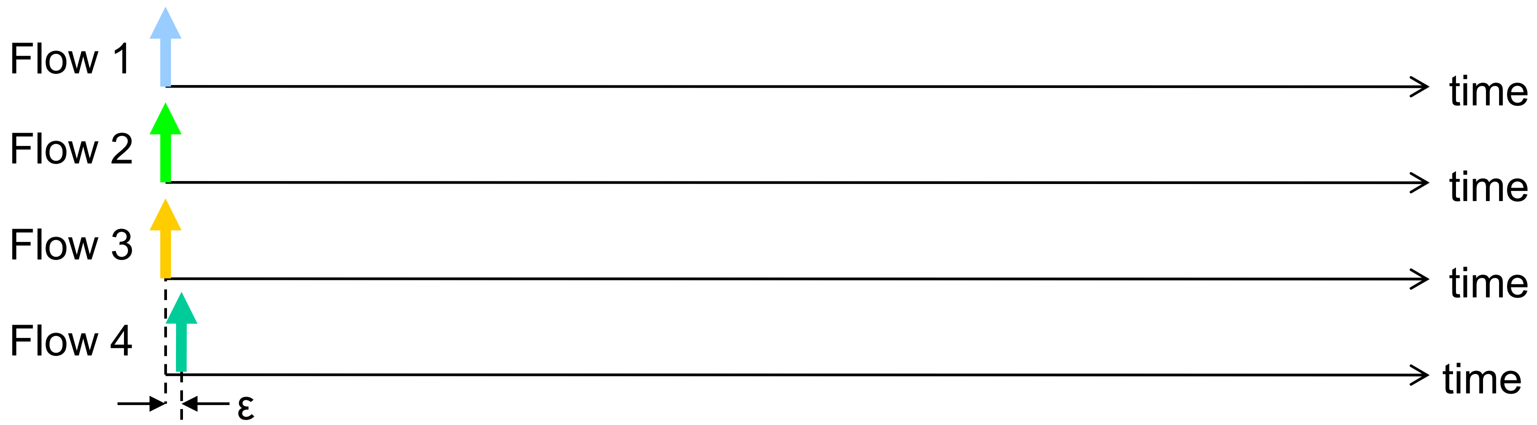
- Four flows, each with weight 1



Solution: Virtual Time

- Key Observation: while the finish times of packets may change when a new packet arrives, the order in which the older packets (those already present in the queue) finish doesn't!
  - Only the order is important for scheduling
- Solution: instead of the packet finish time maintain the round # when a packet finishes (virtual finishing time)
  - Virtual finishing time doesn't change when a packet arrives
- System virtual time V(t) – index of the round in the bit-by-bit round robin scheme

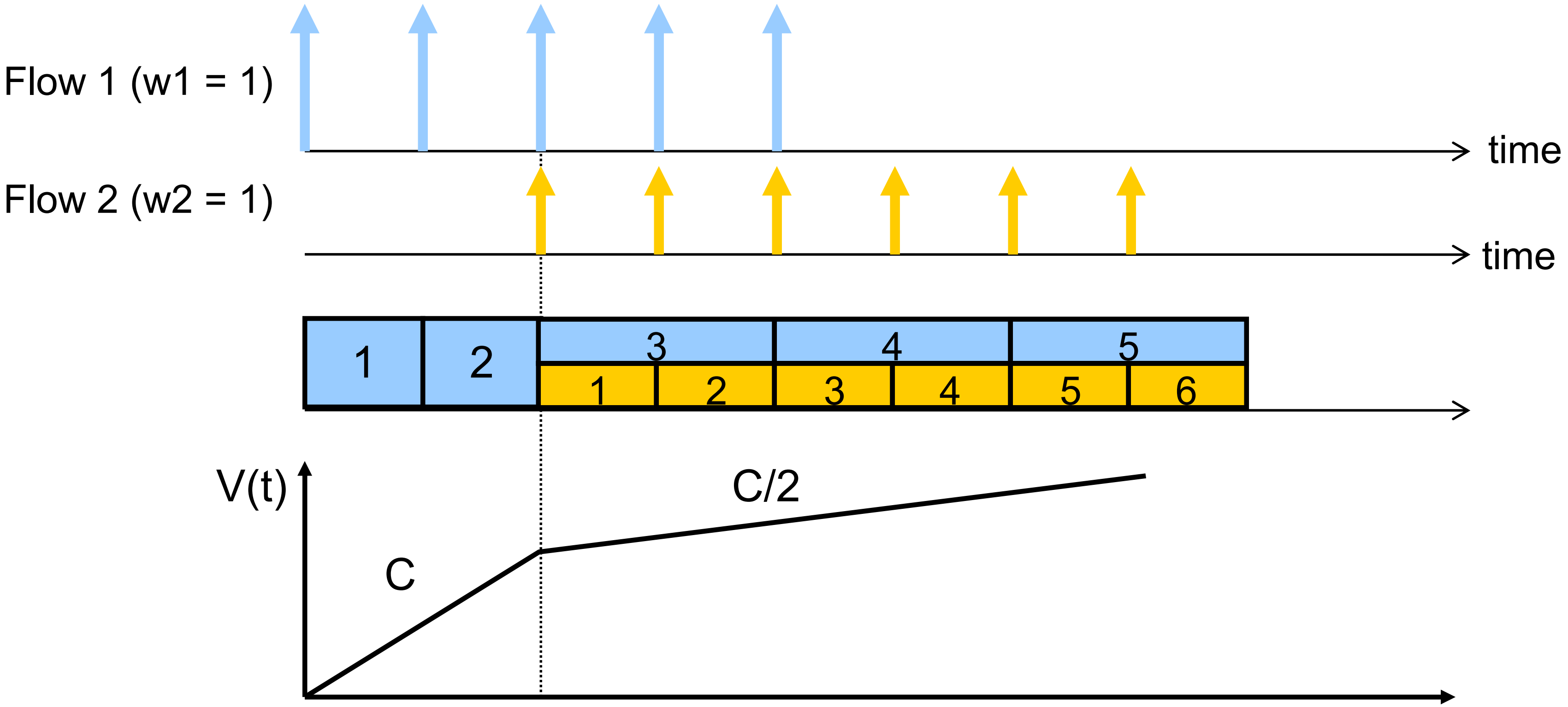
Example



- Suppose each packet is 1000 bits, so takes 1000 rounds to finish
- So, packets of F1, F2, F3 finishes at virtual time 1000
- When packet F4 arrives at virtual time 1 (after one round), the virtual finish time of packet F4 is 1001
- But the virtual finish time of packet F1,2,3 remains 1000
- Finishing order is preserved

System Virtual Time (Round #): V(t)

- V(t) increases inversely proportionally to the sum of the weights of the backlogged flows
- Since round # increases slower when there are more flows to visit each round.



Fair Queueing Implementation

- Define
  - F_i^k - virtual finishing time of packet k of flow i
  - a_i^k - arrival time of packet k of flow i
  - L_i^k - length of packet k of flow i
  - w_i – weight of flow i
- The finishing time of packet $k+1$ of flow i is

$$F_i^{k+1} = \max(V(a_i^{k+1}), F_i^k) + L_i^{k+1} / w_i$$

- Smallest finishing time first scheduling policy

Properties of WFQ

- Guarantee that any packet is transmitted within *packet_length/link_capacity* of its transmission time in the fluid flow system
  - Can be used to provide guaranteed services
- Achieve fair allocation
  - Can be used to protect well-behaved flows against malicious flows